NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

NAVAL ARCHITECTURE ENVIRONMENT: FACILITATING JV2010

by

Thomas H. Augustine

December 1999

Thesis Advisor:	Luqi
Second Reader:	Barbara McBride

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.			
1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE December 1999	3. REPORT TYPE AND DATES COVERED Master's Thesis	
4. TITLE AND SUBTITLE **NAVAL ARCHITECTURE ENVIRONMENT: FACILITATING JV2010**			5. FUNDING NUMBERS
6. AUTHORS Augustine, Thomas H.			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Space and Naval Warfare Systems Center, San Diego 53560 Hull St. San Diego, CA 92152-5001			10. SPONSORING/MONITORING AGENCY REPORT NUMBER
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.			
12a. DISTRIBUTION/AVAILABILITY STATEMENT **APPROVED FOR PUBLIC RELEASE; DISTRIBUTION IS UNLIMITED.**			12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This thesis demonstrates that the C4ISR Framework Version 2.0 requirements can be satisfied with one modern object oriented CASE tool. It provides an alternative scenario-centric approach to architecture development. The combination of scenarios and Unified Modeling Language (UML) semantics is referred to as the Naval Architecture Environment (NAE). Specifically, it recommended the acquisition of Rational Rose.

The NAE combines the best practices of software development with the domain-specific insight contained in the Framework to create an efficient process, supported by a commercial tool and robust semantics, to allow the analysis and design of interoperable C4ISR systems. These are systems that will support Joint Vision 2010's call for Information Superiority.

14. SUBJECT TERMS C4ISR. Architecture. Unified Modeling Language (UML)			15. NUMBER OF PAGES 62
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT SAR

NAVAL ARCHITECTURE ENVIRONMENT: FACILITATING JV2010

Thomas H. Augustine
B.S.E.E., University of Californian, Los Angeles, 1983

Submitted in partial fulfillment of the
Requirements for the degree of

MASTER OF SCIENCE IN SOFTWARE ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
December 1999

Author: [illegible]
Thomas H. Augustine

Approved by: Luqi
Luqi, Thesis Advisor

[illegible]
Barbara McBride, Second Reader

Luqi
Luqi, Chairman
Software Engineering

ABSTRACT

This thesis demonstrates that the C4ISR Framework Version 2.0 requirements can be satisfied with one modern object oriented CASE tool. It provides an alternative scenario-centric approach to architecture development. The combination of scenarios and Unified Modeling Language (UML) semantics is referred to as the Naval Architecture Environment (NAE). Specifically, it recommended the acquisition of Rational Rose.

The NAE combines the best practices of software development with the domain-specific insight contained in the Framework to create an efficient process, supported by a commercial tool and robust semantics, to allow the analysis and design of interoperable C4ISR systems. These are systems that will support Joint Vision 2010's call for Information Superiority.

TABLE OF CONTENTS

LIST OF FIGURES

I. INTRODUCTION

A. PURPOSE

This thesis explores and demonstrates how commercially available object-oriented analysis and design tools can be used to develop C4ISR Architectures that fulfill the essential requirements of the C4ISR Architecture Framework. This process will create internally consistent and maintainable products in a cost-effective COTS environment.

B. MOTIVATION

The Services and Agencies of the Department of Defense (DoD)have been faced with falling Total Obligation Authority (TOA) since the fall of the Berlin Wall. While manning and quality of life have been affected the impact on the acquisition community has been even more severe. As Paul G. Kaminski, undersecretary of defense for acquisition and technology, stated at the Center for Strategic and International Studies Inaugural Conference, Washington, Feb. 27, 1995,

> In response to reduced mean value of the threat the United States has cut end strength by about a third from 1985 levels. But at the same time the increase in variance has caused deployments of U.S. forces to go up by a third. During this adjustment phase we have brought the total defense budget down while maintaining the high state of readiness needed to support increased operational tempos. We have done this by reducing our procurement at a pace that is twice the rate of the overall downturn in total obligation authority. This response is consistent with historical norms. Procurement has been the most volatile component of the budget in a drawdown because it is not necessary to purchase new equipment for a smaller force structure. [1]

The acquisition community is in the midst of a paradigm shift driven by falling TOA. The nature of this shift will be discussed shortly. No longer can the DoD financially afford disparate and redundant systems with similar functions being supplied to different communities by different suppliers with different supply channels. Dollars drive the debate but they are not the only factor. Two other major changes are contributing. First is the drive toward Joint operations that requires interoperability. The second factor discouraging military specific development is the phenomenal growth in the private computing industry. Growth that has reduced the DoD from being a major customer to a rather insignificant one while providing a plethora of options to build upon.

The shift in paradigm is a shift from *program-centric acquisition* to *warfighter-centric acquisition.* The warfighter must be central to the acquisition system because there are not the dollars available to obligate to any cause that does not provide the maximum return in warfighter effectiveness.

The old paradigm was characterized by system "stove-pipes" operated by closed communities. The new paradigm will be characterized by horizontal integration, warfighter optimization across systems, increased importance of Joint service, and the use of Commercial Off- the-Shelf (COTS) systems.

C. ORGANIZATION

1. C4ISR Architectures

This chapter will discuss the benefits of C4ISR Architectures, describe an example product, and discuss its use in the evolving acquisition environment.

C4ISR Architectures are intended to capture warfighter requirements and allocate them efficiently across systems. To accomplish this requires a comprehensive environment of supporting tools and processes.

2. Current C4ISR Architectures Development Environment

This chapter will describe the status of the current C4ISR architectures development environment. It provides a high level view of the components of an architectures development environment.

3. Essentials of the Unified Modeling Language

This chapter discusses the fundamentals of the Unified Modeling Language (UML). It describes modern modeling techniques and mechanisms.

4. Rational Rose

This chapter describes Rational Corporation's Rose modeling product. Along with an overview of the Rose tool, this chapter discusses the advantages and disadvantages of Rational's implementation.

5. Naval Architecture Environment (NAE)

This chapter will describe an augmented development environment for architectures that utilize the underlying capabilities of Rose to provide user support.

6. Recommendations and Conclusions

This chapter recommends areas of additional research and concludes with a summary analysis of NAE.

II. C4ISR ARCHITECTURES

A. THE C4ISR ARCHITECTURE FRAMEWORK

1. The Document

The C4ISR Architecture Working Group, under the sponsorship of ASD(RDT&E), has pursued the necessary conditions for the Services and Agencies of the DoD to acquire interoperable systems. This effort has resulted in, most recently, the promulgation of the *C4ISR Architecture Framework, Version 2* [2].

The development of the Framework was in response to recent government legislation that is placing more emphasis on the need to pursue interoperable, integrated, and cost-effective business practices and capabilities within each organization and across DoD, particularly with respect to information technology. Two legislative acts that impact DoD architecture analysis and integration activities are the Information Technology Management Reform Act (ITMRA), also known as the Clinger-Cohen Act of 1996, and the Government Performance and Results Act of 1993 (GPRA). Together, the ITMRA and GPRA serve to codify the efficiency, interoperability, and leveraging goals being pursued by the Services and Agencies of DoD.

The ITMRA and the GPRA require DoD organizations to measure the performance of existing and planned information systems and to report performance measures on an annual basis. The *C4ISR Architecture Framework* provides uniform methods for describing information systems and their performance in the context of mission and functional effectiveness.

The *Framework* defined three views of an integrated architecture: Operational, Systems, and Technical. It is important to remember that they are intended to be views, not complete and independent models unto themselves.

2. The Operational Architecture View

The operational architecture (OA) view is a description of the tasks and activities, operational elements, and information flows (known as Information Exchange Requirements (IER)) required to accomplish or support a military operation.

The OA view contains descriptions (often graphical) of the operational elements, assigned tasks and activities, and information flows required to support the warfighter. It defines the types of information exchanged, the frequency of exchange, which tasks and activities are supported by the information exchanges, and the nature of information exchanges in sufficient detail to ascertain specific interoperability requirements.

Tenets that apply to the operational architecture view include the following:

The primary purpose of an operational architecture is to define operational elements, activities and tasks, and information exchange requirements.

Operational architectures incorporate doctrine and assigned tasks and activities.

Activities and IERs may cross-organizational boundaries.

Operational architectures are not generally systems-dependent.

Generic activity descriptions are not based on an organizational architecture or force structure.

Operational architectures should clearly identify the time phase(s) covered (e.g., specific years; "as-is" or "to-be").

3. Definition of the Systems Architecture View

The systems architecture (SA) view is a description, including graphics, of systems and interconnections providing for, or supporting, warfighting functions.

For a domain, the SA view shows how multiple systems link and interoperates, and may describe the internal construction and operations of particular systems within the architecture. For the individual system, the SA view includes the physical connection, location, and identification of key nodes (including materiel item nodes), circuits, networks, warfighting platforms, etc., and specifies system and component performance parameters (e.g., mean time between failure, maintainability, availability). The systems architecture view associates physical resources and their performance attributes to the operational view and its requirements per standards defined in the technical architecture.

Tenets that apply to the systems architecture include the following:

The primary purpose of systems architecture is to enable or facilitate operational tasks and activities through the application of physical resources.

Systems architectures map systems with their associated platforms, functions, and characteristics back to the operational architecture.

Systems architectures identify system interfaces and define the connectivities between systems.

Systems architectures define system constraints and bounds of system performance behavior.

Systems architectures are technology-dependent (unlike operational architectures), show how multiple systems within a subject area link and interoperate, and may describe the internals of particular systems.

Systems architectures can support multiple organizations and missions.

Systems architectures should clearly identify the time phase(s) covered.

Systems architectures are based upon and constrained by technical architectures.

4. Definition of the Technical Architecture View

The technical architecture (TA) view is the minimal set of rules governing the arrangement, interaction, and interdependence of system parts or elements, whose purpose is to ensure that a conformant system satisfies a specified set of requirements.

The TA view provides the technical systems-implementation guidelines upon which engineering specifications are based, common building blocks are established, and product lines are developed. The TA view includes a collection of the technical standards, conventions, rules and criteria organized into profile(s) that govern system services, interfaces, and relationships for particular systems architecture views and that relate to particular operational views.

Tenets that apply to the TA view include the following:

TA views are based on associations between operational requirements and their supporting systems, enabling technologies, and appropriate interoperability criteria.

The primary purpose of a TA is to define the set of standards and rules that govern system implementation and system operation.

A TA profile is constructed from an enterprise-wide set of standards and design rules for specific standards contained in the Joint Technical Architecture[3] and other applicable standards documents.

The TA standards and criteria should reflect multiple information system implementation paradigms.

TA profiles account for the requirements of multi-platform and network interconnections among all systems that produce, use, or exchange information electronically for a specifically bounded architecture configuration.

Technical architectures must accommodate new technology, evolving standards, and the phasing out of old technology.

Technical architectures should be driven by commercial standards and direction.

B. JOINT VISION 2010

A key component of Operational Architecture is a vision document that outlines the mission that the architecture will fulfill.

The Chairman of the Joint Chiefs of Staff has outlined his future vision of twentieth century warfare in a document titled *Joint Vision 2010*[4]. Figure 1 captures the concepts of JV 2010 in one diagram.

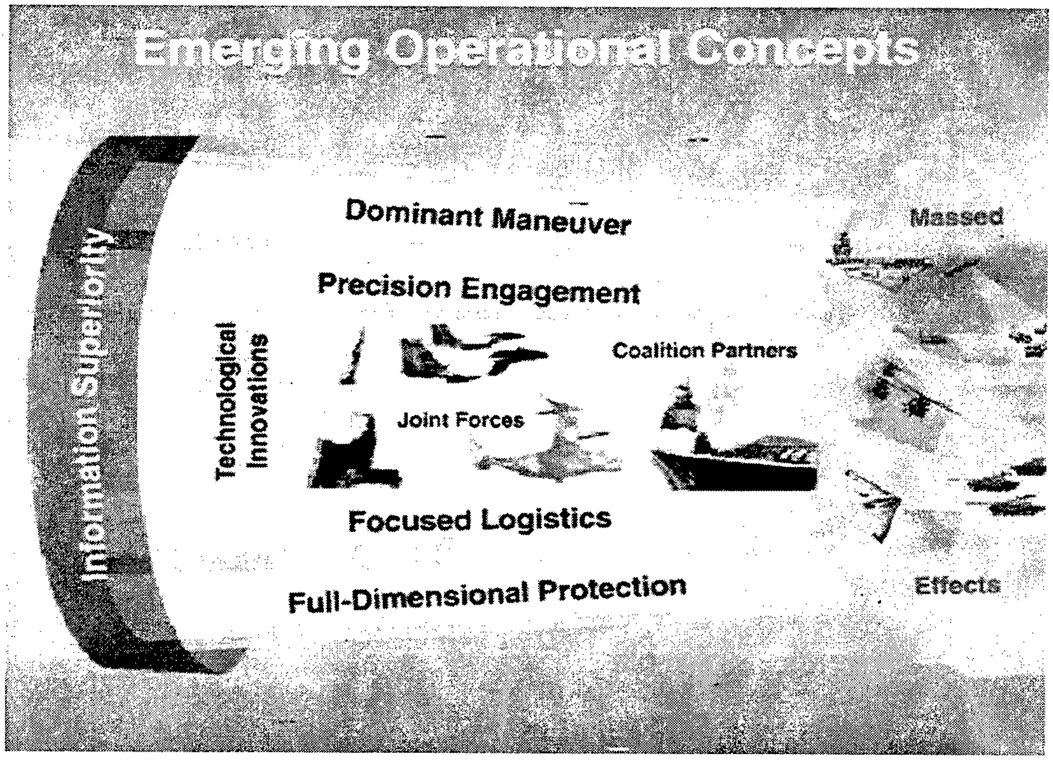


Figure 1: Tenets of JV 2010

JV 2010 is based upon the tenets of dominant maneuver, precision engagement, focused logistics, and full-dimensional protection. Note that the four tenets are all encompassed by *information superiority*. In an era of diminishing force structure, the success of our nation's military defense is tied to its ability to do more with less - as mentioned earlier. Information superiority is the force multiplier that enables a smaller, more agile, technologically superior force to succeed in battle. The doctrine that outlines how the Department of Defense will achieve information superiority is published in Joint Pub 6[5].

C. C4I FOR THE WARRIOR: JOINT PUB 6

Command, Control, Communications, Computers, and Intelligence (C4I) encompasses the procedures that a commander employs to direct his forces, as well as the policies to be used to communicate information between them. *C4I For the Warrior* is the vision for future command and control systems. In this case, the *Warrior* refers to the warfighting commander. In order to accomplish his mission, the warrior needs a fused, current, and accurate representation of the battlespace along with the ability to coordinate with, respond to, and order all his forces. The Joint Staff concisely defines the importance of command and control in the following quote from its doctrine:

War is a process that pits the opposing wills of two commanders against each other. Great victories of military forces are often attributed to superior firepower, mobility, or logistics. In actuality, it often is the commander who makes good decisions and executes these decisions at a superior tempo who leads his forces to victory.

Therefore, victory demands that commanders effectively link decision making to execution through the concept of command and control. Warfare will continue to evolve and command and control processes, organization, and supporting systems will continue to change, but the basic concept of command and control will remain the key to the decisive application of combat power. More than ever before, a command and control system is crucial to success and must support shorter decision cycles and instantaneous flexibility across vast distances of time and space. [6]

Clearly, if we are to achieve information dominance, we need a coherent and comprehensive C4ISR architecture that allows us to intelligently and currently utilize COTS capabilities more rapidly than our adversary. Command of joint forces in war is an intense, competitive and stressful process. The joint force commander is not only faced with making life and death decisions in complex situations but must do this, in limited time, in an environment of uncertainty. Command is as much a problem of information system management as it is of carrying out difficult and complex warfighting tasks.

Command, control, communications, computers and Intelligence (C4I) systems supporting US military forces must have the capability to rapidly ***adapt*** to the demands of the commanders who use them, as well as the environment in which they are used. They must make important information available, provide it where needed, and ensure that it gets there not only in a timely manner, but also in a format that is usable by the receiver. In short, there are key information exchange requirements that can be architecturally captured and must be supported. The Joint Chiefs of Staff summarize the goal of C4I systems as follows: "The fundamental objective of C4I systems is to get the critical and relevant information to the right place in time to allow forces to seize on opportunity and meet the objectives across the range of military operations."[7]

C4ISR systems are extensions of the natural needs and actions of the commander. And architectures should naturally capture that relationship.

D. SUMMARY

C4ISR systems and the explosive growth of computing have irrevocably changed the paradigm by which systems of all types are acquired. Long gone are the days specialized systems for closed communities. The ability to present previously unimaginable amounts of information inexpensively has affected the manner in which Commanders will employ information technology assets. Joint Vision 2010 requires information superiority for success in battle. C4ISR architectures facilitate that objective.

THIS PAGE INTENTIONALLY LEFT BLANK

III. CURRENT C4ISR ARCHITECTURES DEVELOPMENT ENVIRONMENT

A. PURPOSE

The purpose of this section is to provide the reader with a sense of the current development environment. This will include a sense of the lifecycle costs associated with the current methods as well as their semantic limitations.

The current development environment supports the development of the essential and supporting architecture products (see **Figure 2: Framework Products**) through an ad-hoc collection on non-interoperable software tools.

The Information Exchange Requirements are stored in a relational database. Activity diagrams are developed in BPWin. Top level concept diagrams are developed in Microsoft Powerpoint. Various types of entity relationship graphics are produced using different tools. Command hierarchy graphics may be developed in Microsoft PowerPoint. Database schemas are visualized by a tool unique to the database being used or perhaps ERWin.

These tools were not designed to interoperate and do not support a coherent information architecture. As such it has been the author's experience that expert developers are necessary to develop coherent architectures. That is a key criticism since a measure of effectiveness of any modeling

Applicable Architecture View	Product Reference	Architecture Product	Essential or Supporting	General Nature
All Views (Context)	AV-1	***Overview and Summary Information***	**Essential**	Scope, purpose, intended users, environment depicted,analytical findings, if applicable *(4.2.1.1)*
All Views (Terms)	AV-2	***Integrated Dictionary***	**Essential**	Definitions of all terms used in all products *(4.2.1.2)*
Operational	OV-1	***High-level Operational Concept Graphic***	**Essential**	High-level graphical description of operational concept (high-level organizations, missions, geographic configuration, connectivity, etc.) *(4.2.1.3)*
Operational	OV-2	***Operational Node Connectivity Description***	**Essential**	Operational nodes, activities performed at each node, connectivities & information flow between nodes *(4.2.1.4)*
Operational	OV-3	***Operational Information Exchange Matrix***	**Essential**	Information exchanged between nodes and the relevant attributes of that exchange such as media, quality, quantity, and the level of interoperability required. *(4.2.1.5)*
Operational	OV-4	***Command Relationships Chart***	Supporting	Command, control, coordination relationships among organizations *(4.2.2.1)*
Operational	OV-5	***Activity Model***	Supporting	Activities, relationships among activities, I/Os, constraints (e.g., policy, guidance), and mechanisms that perform those activities. In addition to showing mechanisms, overlays can show other pertinent information. *(4.2.2.2)*
Operational	OV-6a	***Operational Rules Model***	Supporting	One of the three products used to describe operational activity sequence and timing that identifies the business rules that constrain the operation *(4.2.2.3.1)*
Operational	OV-6b	***Operational State Transition Description***	Supporting	One of the three products used to describe operational activity sequence and timing that identifies responses of a business process to events *(4.2.2.3.2)*
Operational	OV-6c	***Operational Event/Trace Description***	Supporting	One of the three products used to describe operational activity sequence and timing that traces the actions in a scenario or critical sequence of events *(4.2.2.3.3)*
Operational	OV-7	***Logical Data Model***	Supporting	Documentation of the data requirements and structural business process rules of the Operational View. *(4.2.2.4)*
Systems	SV-1	***System Interface Description***	**Essential**	Identification of systems and system components and their interfaces, within and between nodes *(4.2.1.6)*
Systems	SV-2	***Systems Communications Description***	Supporting	Physical nodes and their related communications laydowns *(4.2.2.5)*
Systems	SV-3	***Systems² Matrix***	Supporting	Relationships among systems in a given architecture; can be designed to show relationships of interest, e.g., system-type interfaces, planned vs. existing interfaces, etc. *(4.2.2.6)*
Systems	SV-4	***Systems Functionality Description***	Supporting	Functions performed by systems and the information flow among system functions *(4.2.2.7)*
Systems	SV-5	***Operational Activity to System Function Traceability Matrix***	Supporting	Mapping of system functions back to operational activities *(4.2.2.8)*
Systems	SV-6	***System Information Exchange Matrix***	Supporting	Detailing of information exchanges among system elements, applications and H/W allocated to system elements *(4.2.2.9)*
Systems	SV-7	***System Performance Parameters Matrix***	Supporting	Performance characteristics of each system(s) hardware and software elements, for the appropriate timeframe(s) *(4.2.2.10)*
Systems	SV-8	***System Evolution Description***	Supporting	Planned incremental steps toward migrating a suite of systems to a more efficient suite, or toward evolving a current system to a future implementation *(4.2.2.11)*
Systems	SV-9	***System Technology Forecast***	Supporting	Emerging technologies and software/hardware products that are expected to be available in a given set of timeframes, and that will affect future development of the architecture *(4.2.2.12)*
Systems	SV-10a	***Systems Rules Model***	Supporting	One of three products used to describe systems activity sequence and timing – Constraints that are imposed on systems functionality due to some aspect of systems design or implementation *(4.2.2.13.1)*
Systems	SV- 10b	***Systems State Transition Description***	Supporting	One of three products used to describe systems activity sequence and timing – Responses of a system to events *(4.2.2.13.2)*
Systems	SV -10c	***Systems Event/Trace Description***	Supporting	One of three products used to describe systems activity sequence and timing – System-specific refinements of critical sequences of events described in the operational view *(4.2.2.13.3)*
Systems	SV-11	***Physical Data Model***	Supporting	Physical implementation of the information of the Logical Data Model, e.g., message formats, file structures, physical schema *(4.2.2.14)*
Technical	TV-1	***Technical Architecture Profile***	**Essential**	Extraction of standards that apply to the given architecture *(4.2.1.7)*
Technical	TV-2	***Standards Technology Forecast***	Supporting	Description of emerging standards that are expected to apply to the given architecture, within an appropriate set of timeframes *(4.2.2.15)*

Figure 2: Framework Products

system is its ability to aid the novice in achieving expert level understanding.

B. BENEFITS AND SHORTFALLS

The current "system" benefits from minimizing the up-front lifecycle costs:

1) User training.
2) Tool acquisition.
3) Tool maintenance costs.

The current "system" experiences shortfalls in the down-stream lifecycle costs:

1) Architecture maintenance.
2) Architecture revision.
3) Architecture adaptation.
4) Architecture correctness (lower return on investment).

A result of the current system, which creates the model at the database level, is that the developer is responsible for ensuring database correctness. This is not a minor item since databases (specifically, Microsoft Access) are much more "user-friendly" appearing than they are in practice to use *correctly*. An example is shown (see Figure 3 : Violation of Database Integrity).

As discussed previously the development methodology emphasized the population of relational database tables. These tables would then be queried to produce actual products. The development methodology also proposed a chain of

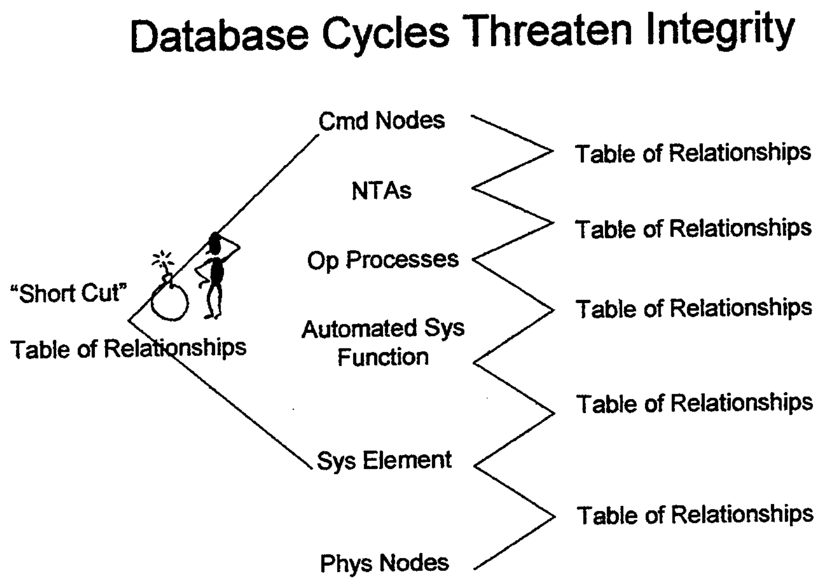


Figure 3 : Violation of Database Integrity

Relationships, shown in Figure 3 : Violation of Database Integrity, which when populated and subsequently queried would provide the mapping of system elements to command nodes. This query would identify which command node required which system element.

However, once the Command Node, NTA (tasks), Operational Processes, Automated System Function, and System Element were populated and a query run to produce the table of system elements to command nodes the resulting table was fully populated; i.e. all system elements were to be on all command nodes! This was not the "correct answer".

Due to lack of time and resources it was not considered feasible to revisit the four table of relationships that were used in the query. Instead a new table, annotated as a "Short Cut" in Figure 3, was created. This new table contained the "correct answer."

In his seminal paper on relational databases, "A Relational Model of Data for Large Shared Data Banks,"[8] when referring to what is now know as the first normal form E.F. Codd writes:

If normalization as described above is to be applicable, the unnormalized collection of relations must satisfy the following conditions:

1. The graph of interrelationships of the nonsimple domains is a collection of trees
2. No primary key has a component domain which is nonsimple.

The writer knows of no application which would require any relaxation of these conditions. Further operations of a normalizing kind are possible. These are not discussed in this paper.

The insertion of the table that directly related system elements to command nodes violated Codd's first condition. No longer is the schema is a tree. It has become cyclical.

This violation of relational database integrity will result in ongoing costs and confusion. It is an excellent example of why analysis and design should not be executed at the database level.

An additional consideration when using a relational database to derive answers to queries is that each relation should be a true statement. If that is not the case then the outcome of projections is apt to be misleading.

C. SUMMARY

Building C4ISR Architectures at the database level is not a cost effective approach. Not only is it expensive (from a lifecycle perspective) it is also *dangerous*. When novices are empowered to *think* they are experts unnecessary risks are taken. That is the true definition of danger.

A more robust environment is needed which abstracts the database and supports the desire of the user to have multiple *views* of one model.

IV. ESSENTIALS OF THE UNIFIED MODELING LANGUAGE

A. THE ADVANTAGES OF OBJECTS

The primary advantages of object-oriented development, as compared to the structured analysis (data flow diagrams, functional decomposition, and activity diagrams) methods of current architecture efforts, are:

1. Consistency of Model Views

Data flow diagrams do not map well into implementations, which are designed using a different paradigm. This problem appears in the architecture Framework products, as currently produced, because the mapping from operational architecture products to system architecture products is not clear - certainly not automated.

2. Improved Problem Domain Abstraction

Object technology binds data and functionality together to provide a more robust abstraction. In the structured approach functions are separated from data. This dichotomy appears in the creation of activity models in BPWin and storing of IERs in a separate database in operational architectures.

The ability to tie data and functionality provides an abstraction that more closely matches the world as we experience it. A sensor (for example) can be modeled as an object with certain methods (functions) and related attributes (data) - the relation of methods to attributes is maintained.

3. Improved Stability with Functional Changes

The area of greatest of change in most projects comes from customers changing their minds or, after seeing what has been developed, deciding they wanted something else. That "something else" is usually a change to functionality. It is not a change to the customer (or "actor") set, it is not a change to fundamental elements ("objects"), and it is usually a change to the how the system elements interact with the customer. In an object-oriented model these changes are isolated to the affected object's methods.

4. Improved Model Facilities for Reuse

Reusing functionally decomposed components is difficult without rigid interface specifications that are uncommon in software development (beyond mathematical, logical, and relational functions). This problem is critical at the architecture level, which, of course, is at a much higher level of abstraction than the mathematical, logical, or relational level.

Object technology uses generalization and refinement to build hierarchies that can be adopted at each level. Generalization, also known as inheritance, allows reuse by adding and extending existing models without modifying their underlying structure. Refinement builds upon incomplete specifications, or templates, to allow more specific implementations (and allow for changes that are implementation specific).

5. Improved Scalability

As an article of faith Object Oriented developers believe "Objects do it better!" There are some interesting possible connections in the world of DoD acquisition to believe this is true. A key element in the system development life cycle is modeling and simulation (frequently referred to as "M&S"). M&S is used not only in prototyping, development, and testing, but also in embedded training. As this becomes a larger cost driver for programs more effort will be expended in controlling that cost through re-use. A likely vehicle towards that end, currently used for developing federations of simulations, is the High Level Architecture (HLA)[9] sponsored by the Defense Modeling and Simulation Office. The HLA is fundamentally object based. At the other side of the life cycle, requirements, there are many interesting COTS applications that support the round-trip evolution of requirements documents and UML use-case diagrams.[10]

The greatest stumbling block to scalability is when different parts of the acquiring and developing organization use incompatible methodologies that create discontinuities at the organizational interfaces. These discontinuities will add additional costs over the project's life cycle. Object oriented technology offers a methodology that can span the lifecycle and scale to accommodate growth and change.

6. Better Support for Reliability

Well defined interfaces eliminate inconsistencies that otherwise lead to errors and failures. Superior reuse capabilities allow more thoroughly tested components to be used. Objects' well defined interfaces and reuse capabilities are major plusses for reliability.

7. Inherent Support for Concurrency

Concurrency is a concept that is missing from architectures as currently developed. It is also missing from structured methods which do not have mechanisms to capture concurrency, manage tasks, or synchronize tasks.

It would be argued that at the architectural level those issues are not important (although certainly a shooting war is concurrent!). That is a mistake, because it guarantees that your architecture will not be scalable.

B. OMG STANDARD

The Object Management Group, an industry consortium, adopted UML as the object modeling standard. The OMG's mission, in part, is:

> The OMG was formed to create a component-based software marketplace by hastening the introduction of standardized object software. The organization's charter includes the establishment of industry guidelines and detailed object management specifications to provide a common framework for application development. Conformance to these specifications will make it possible to develop a heterogeneous computing environment across all major hardware platforms and operating systems.[11]

The OMG is structured into three major bodies, the Platform Technology Committee (PTC), the Domain Technology Committee (DTC) and the Architecture Board. The Architecture Board is responsible for UML and they describe the scope as follows:

1. Scope of the OMG-UML

> The Unified Modeling Language (UML) is a language for specifying, constructing, visualizing, and documenting the artifacts of a software-intensive system. First and foremost, the Unified Modeling Language fuses the concepts of Booch, OMT and OOSE. The result is a single, common, and widely usable modeling language for users of these and other methods.
> Second, the Unified Modeling Language pushes the envelope of what can be done with existing methods. As an example, the UML authors targeted the modeling of concurrent, distributed systems to assure that the UML adequately addresses these domains.
> Third, the Unified Modeling Language focuses on a standard modeling language, not a standard

process. Although the UML must be applied in the context of a process, experience has shown that different organizations and problem domains require different processes. (For example, the development process for shrink-wrapped software is an interesting one, but building shrink-wrapped software is vastly different from building hard-real-time avionics systems upon which lives depend.) Therefore, the efforts concentrated first on a common metamodel (which unifies semantics) and second on a common notation (which provides a human rendering of these semantics). The UML authors promote a development process that is *use-case driven, architecture centric,* and *iterative and incremental.*

2. Outside The Scope of the UML

While the UML aims to simplify and standardize modeling it is not an all encompassing language. This gives it the flexibility to be used to design a variety of systems over a wide spectrum of industries. Some major areas outside of the scope of the UML include:

a) Programming Languages

The UML, a visual *modeling* language, is not intended to be a visual *programming* language, in the sense of having all the necessary visual and semantic support to replace programming languages. The UML is a language for visualizing, specifying, constructing, and documenting the artifacts of a software-intensive system, but it does draw the line as you move toward code. The UML does have a tight mapping to a family of OO languages, so that you can get the best of both worlds

b) Tools

Standardizing a language is necessarily the foundation for tools and process. The primary goal of the OMG RFP was to enable tool interoperability. However, tools and their interoperability are very dependent on a solid semantic and notation definition, such as the UML provides. The UML defines a *semantic* metamodel, not an tool *interface, storage,* or *run-time* model, although these should be fairly close to one another.

c) *Process*

Many organizations will use the UML as a common language for its project artifacts, but will use the same UML diagram types in the context of different processes. The UML is intentionally process independent, and defining a standard process was not a goal of the UML or OMG's RFP.[12]

C. ORTHOGONAL CHARACTERISTICS OF OBJECTS

A model captures the following three orthogonal (non-redundant) information types (see Appendix A: Example for examples):

1. Dynamic:

Time dependent behavior, essential for capturing concurrency, is captured in state charts and scenarios.

2. Functional:

The functions that are performed by the system are captured in Use Case diagrams and performed by object methods.

3. Entitiy:

The relationships between system entities are captured in class diagrams. Implementation details can be captured in component diagrams and package diagrams.

D. UML MODELS, VIEWS, AND DIAGRAMS

This section is not meant to be a tutorial on UML; there are many *books* on that subject, but rather an overview of how UML products can support the architecture process.

The primary UML vehicle for gathering subject matter expert (SME) input is the use case diagram. Use cases are scenarios. They may be nested hierarchically. The top level use case shows all the scenarios (or sets of scenarios) for which the external environment interacts with the system in question. Interacting with the system in each use case are actors (identified by a stick man stereotype). Underlying each use case can be additional use cases or, at the leaf level, state and activity diagrams. The state and activity diagrams are used to capture the scenario's business model.

The UML offers two semantically identical but visually distinct forms of interaction diagrams: the collaboration diagram and the sequence diagram. Both diagrams would be useful for showing operational node connectivity. They show the flow of messages between objects with respect to time. A collaboration diagram has sequentially numbered messages and allows the objects to be spatially distributed. A sequence diagram is a waterfall style diagram with the objects listed horizontally across the top and time progressing down in the vertical direction.

The sequence diagram would also be useful for collecting architectural information exchange requirements that traditionally have been shown in a matrix and therefore are more easily recognized in the waterfall format. The sequence diagram also allows an advanced notation to be used called "focus of control (FOC)". FOC shows the period of time during which an object is performing an action, either directly or through an underlying procedure. A source FOC will affect a destination FOC such that if it is moved the destination will be as well.

The transition from operational to system analysis is a common architecture problem. UML offers the deployment diagram, which integrates the software and hardware aspects of a system. UML packages represent logical grouping of model entities, generally related classes that would eventually be implemented in a library or configuration item. UML components represent a software module (source code, binary code, executable, etc.) with a well-defined interface. A UML task is a component and it has its own thread of control. Components can be aggregated into packages. In a deployment diagram packages and/or components can be allocated to processors. A processor is a physical entity with computing power. A deployment diagram can also show UML devices that are physical entities without computing power. And the beauty is that the relationships between the devices and processors can be annotated with stereotypes to show the physical connections (i.e., "RS-232", "TCP-IP", or any other meaningful phrase).

E. THE MODERN CASE TOOL

The modern Computer-Aided Software Engineering (CASE) tool is the product of many years of innovation in a wide variety of fields. They are in fact complex software programs themselves, which incorporate the latest in graphical user interface (GUI) designs, database support, and compiler development. A brief history of contributors, which are especially relevant in the context of DoD architectures, is shown in Figure 4 : CASE Tool History.
As mentioned earlier E.F. Codd's relational model paper proved to be seminal in the development of relational databases. This work has been incorporated into CASE tools. Rational Rose stores its models in a relational database.

The Air Force took a leading role in the 70's and 80's in the study of business process modeling. Unfortunately over time the processes that they developed became too document focused and were not used widely outside of DOD. However, the focus on capturing SME input can be traced to IDEF 1 and 1X.

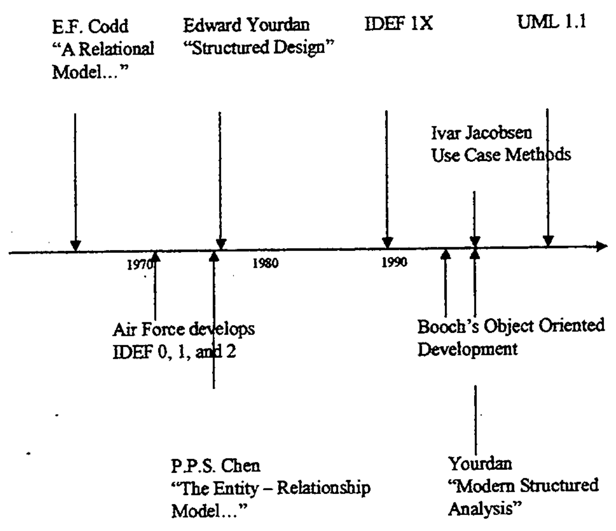


Figure 4 : CASE Tool History

Edward Yourdan has contributed many concepts to the development of modern modeling. As mentioned earlier his concept of "balancing" is now de-rigueur in any CASE tool.

Chen's contribution of entity relationship modeling has influenced not only the analysis and specification of relational databases but also is an underlying concept of classes that are key to object oriented methods.

Grady Booch's contributed one of the earliest object oriented development methods. This built upon the earlier concepts of entity modeling, balanced models, business modeling, and added the concepts of encapsulation and inheritance.

Jacobsen’s use case analysis added a human face to object techniques by providing an interface that SME could easily relate to. Use cases added the ability to capture scenarios and build a system that supports those scenarios.

Finally, when the OMG approved UML v1.1 the computer industry finally agreed upon a semantic standard. This put the “model wars” behind most users and allows the market power of standardization to move to the fore. Now multiple vendors can provide solutions and users may not be locked into a proprietary tool.

F. SUMMARY

UML, as supported by modern CASE tools, captures the three aspects of modeling (dynamic, functional, and entity) that are important to DoD architecture. The relationships between the views and the underlying integrated model are maintained ("balanced" in Yourdan's parlance). The result is an internally consistent, maintainable, and cost-effective COTS environment.

V. NAVAL ARCHITECTURE ENVIRONMENT

The Naval Architecture Environment (NAE) will combine a semantic language (UML), with automated support (Rational Rose), and a process (the NAE process) to provide an internally consistent, maintainable, and cost-effective COTS environment.

A. ASSUMPTIONS

The NAE supports the C4ISR acquisition guidance function of the Chief Engineers office. As such it will support the requirements and top-level architecture efforts. To support verification and validation of architectures it adopts the use of UML, which is consistent with DMSO's HLA. Further research will be required to specify the nature of that interaction.

B. UML: ENSURING CONSISTENCY WITHIN THE FRAMEWORK PRODUCTS

Ed Yourdan says, "A structured specification in which all the modeling tools have been cross-checked against each other for consistency is said to be *balanced.*"[13]

The model views and their products, as produced to date, are not identical to those described by Yourdon or other authors. However, there are strong parallels with the crosschecking requirements that have been identified.

UML builds upon the structured analysis and design legacy to provide extensive crosschecking and enforcement mechanisms. UML 1.1, the current standard, has a rigorous, although not totally formal, language description to provide detailed rules for static and dynamic semantics.

UML 1.1 explains the difference between static and dynamic semantics in the following quotation:

> The static semantics of a language define how an instance of a construct should be connected to other instances to be meaningful, and the dynamic semantics define the meaning of a well-formed construct. The meaning of a description written in the language is defined only if the description is well formed (i.e., if it fulfills the rules defined in the static semantics).[14]

A useful example of the difference between static and dynamic semantics is the definition of class and inheritance. The concept of class, an instance of meta-class, is a basic construct in object-oriented paradigms. The concept of inheritance defines how a class is created from its constituent members. The definition of class is given as a static semantic definition using Object Constraint Language (OCL) and repeated in natural language and the dynamic semantics of inheritance are described in natural language.

The rigor that has been applied to the development of UML can be leveraged to assist in creating models, and views of models, that are internally consistent.

C. RATIONAL ROSE: MARKET LEADING UML CASE TOOL

Rational Software's competitive advantage in the crowded CASE tool market, which has resulted in their market leading position, has been to bring the "gurus" together. The methodologists brought together at Rational are:

1. Jim Rumbaugh

Rumbaugh is best known as a principal developer of the Object Modeling Technique (OMT). OMT presented an object oriented modeling technique that covered the three aspects mentioned earlier (entity (known as the object model), dynamic, and functional). In an OMT model the most important model aspect is the object model. This characteristic distinguishes it from structured analysis techniques that focus on the functional model.

2. Grady Booch

Booch is best known as a principal developer of the Booch Method. The Booch Method was generally similar to OMT; in fact, some tools support either notation. The existence of competing notations added mostly to the level of confusion.

3. Ivar Jacobsen

Jacobsen developed "Use Cases" as a mechanism to discover objects, identify user interface interactions, identify user types, and capture requirements. Rather than publishing or developing commercial tools his work was developed primarily for Ericson, the Swedish electronics giant, and his own consulting firm, Objectory. His process, Object Oriented System Engineering (OOSE), was a use-case oriented approach that provided support for business engineering and requirements analysis.

D. NAE PROCESS

The NAE process supports the development of the Framework's Essential products (see Figure 5: UML Support for NAE Process). The NAE process has a different focus however. While the Framework emphasizes the development of Information Exchange Requirements (IER) the NAE process emphasizes the development of scenarios.

During the development of Framework IERs the subject matter experts (SME) will usually talk in terms of scenarios. For instance, a warfare commander will describe the communications that occur between his staff, other warfare commander's staff, and the composite warfare commander during the replanning process when there is a contention for resources. Each exchange will generate an IER that occurs in the context of the SME's scenario. Once the IERs are collected and stored in OV-3 the scenario has been lost if each is stored as one row in a table in a relational database (the typical implementation of OV-3).

Capturing that exchange in a Scenario Diagram will create the IERs while maintaining their context (a Scenario Diagram is an instance of Use Case) and their temporal

Applicable Architecture View	**Product Reference**	**Architecture Product**	**Essential or Supporting**	**UML Diagram**
All Views	AV-1	Operational Summary Information	Essential	Main Use Case Diagram Documentation
All Views	AV-2	Integrated Dictionary	Essential	Rose Model Database
Operational	OV-1	High-Level Operational Concept Graphic	Essential	Main Use Case Diagram or custom graphic
Operational	OV-2	Operational Node Connectivity Graphic	Essential	Activity, Collaboration, and/or Deployment Diagram

Figure 5: UML Support for NAE Process

Applicable Architecture View	**Product Reference**	**Architecture Product**	**Essential or Supporting**	**UML Diagram**
Operational	OV-3	Operational Information Exchange Matrix	Essential	Scenario-type diagram (Sequence or Collaboration)
Systems	SV-1	System Interface Description	Essential	Implementation -type diagram (Component, Deployment, and/or Package)
Technical	TV-1	Technical Architecture Profile	Essential	Implementation -type diagram Stereotypes

Figure 5: UML Support for NAE Process (continued)

relationship as well.

The current release of Rational Rose, 98i, does not support all the possible UML syntax. One area where this thesis revealed a shortcoming is in the Deployment Diagram (see Appendix A). The example Deployment Diagram in Appendix A shows the full syntax described by the standard.

A Rose deployment diagram shows processors, devices, and connections. Each model contains a single deployment diagram that shows the connections between its processors and devices, and the allocation of its processes to processors. Full support for the Deployment Diagram syntax would include the presentation of packages and tasks as well. These are shown in the example in Appendix A. The Deployment Diagram tends to stand alone without these elements.

On the plus side Rose offers (depending on level purchased) various means to extend its capabilities and interface with other vendor's products. In particular, the ability to leverage its underlying relational database to interact with other products offers the C4ISR architecture community an opportunity to continue to derive value from the extensive relational database tables created to date.

Rose offers an extensibility interface called REI. Rational has already taken advantage of this extensibility to extend Rose 98's capabilities. The REI has enabled Rational to integrate LogicWorks' ERwin product to provide database schema and Data Description Language (DDL) generation capabilities in the Rose 98 Enterprise Edition.

Rose also offers round-trip re-engineering with the Oracle 8 database, which enables analysts to model their business processes and generate object-relational schemas for Oracle8. In addition, developers can use Rational Rose to extract relational schemas from an existing Oracle database and create object views for building an object-oriented application architecture.

E. SUMMARY

The NAE combines the best practices of software development with the domain-specific insight contained in the Framework to create an efficient process, supported by a commercial tool and robust semantics, to allow the analysis and design of interoperable C4ISR systems. These are systems that will support Joint Vision 2010's call for Information Superiority.

VI. RECOMMENDATIONS AND CONCLUSIONS

This thesis has demonstrated that the requirements of Framework V 2.0 can be satisfied with one modern object oriented CASE tool. Specifically, it recommended the acquisition of Rational Rose.

A. RECOMMENDATIONS

The need to efficiently acquire, analyze, and design C4ISR systems is not limited to the United States. A similar effort has been located in Norway[15] and others are probably underway. It is recommended that CISA sponsor a symposium to solicit input for version 3 of the Framework.

It is also recommended that the NAE be applied to an actual program so that practical experience can be gained. The author expects this to occur in Fiscal Year 2000.

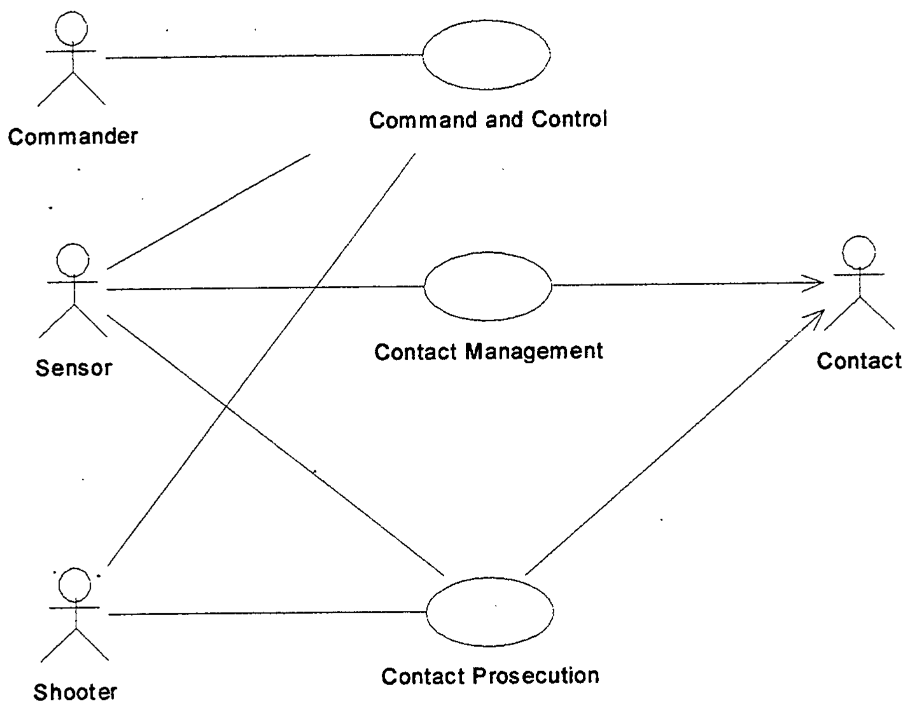
B. SUMMARY AND CONCLUSIONS

When the NAE is fully implemented it can help achieve this internally consistent, maintainable, and cost-effective COTS environment.

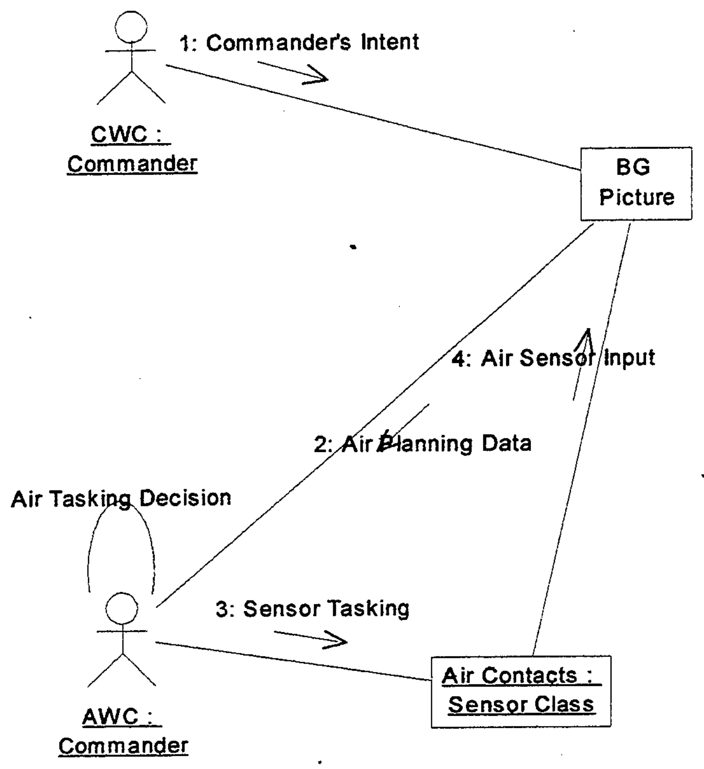
THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A: EXAMPLE FRAMEWORK PRODUCTS USING UML

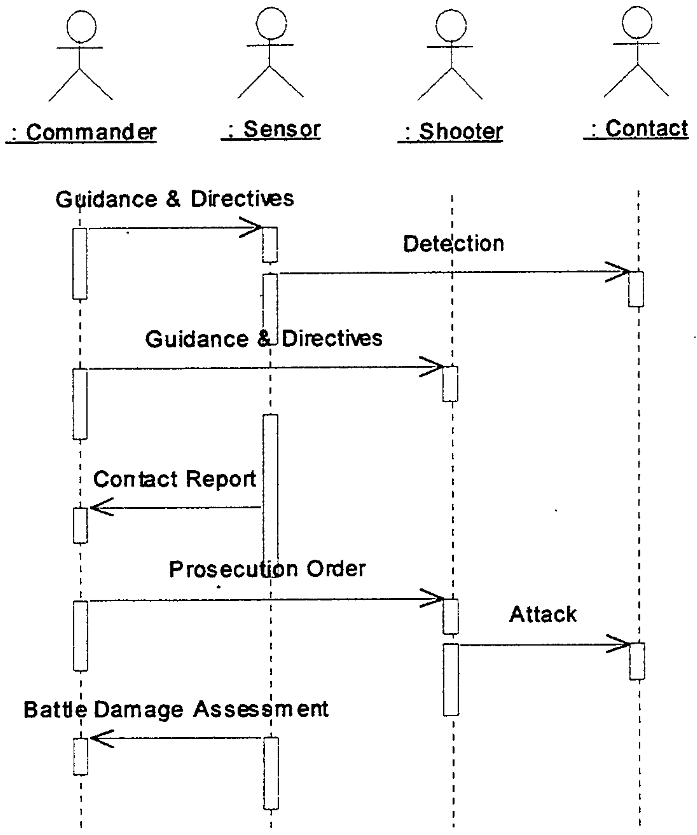
1. OV-1, High-Level Operational Concept Graphic (Use Case Diagram)



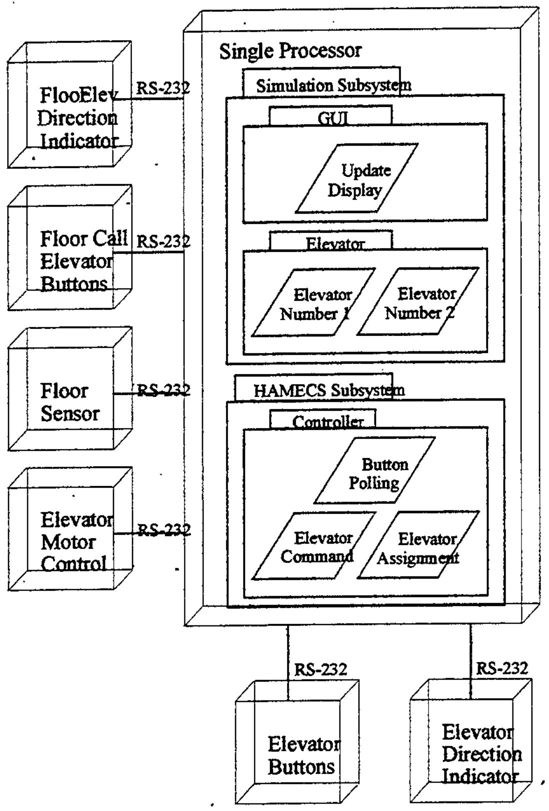
2. OV-2, Operational Node Connectivity Graphic (Collaboration Diagram)



3. OV-3, Operational Information Exchange Matrix (Sequence Diagram)



4. SV-1, System Interface Description (Deployment)



LIST OF REFERENCES

[1] Paul G. Kaminski, Undersecretary of Defense for Acquisition and Technology, stated at the Center for Strategic and International Studies Inaugural Conference, Washington, Feb. 27, 1995

[2] C4ISR Architecture Framework, Version 2, C4ISR Architecture Working Group, 1997.

[3] Available on-line at: http://www-jta.itsi.disa.mil/

[4] Available on-line at http://www.dtic.mil/doctrine/jv2010/jvpub.htm

[5] Available on-line at: http://www.dtic.mil/doctrine/jel/new pubs/jp6 0.pdf

[6] JP 6

[7] JP 6

[8] *Communications of the ACM*, Vol. 13, No. 6, June 1970, pp. 377-387 The first of two sections is available on-line at: http://www.acm.org/classics/nov95/

[9] On-line information available at: http://hla.dmso.mil/

[10] On-line information for Rational's tool available at: http://www.rational.com

[11] Available on-line at: http://www.omg.org/omg/background.html

[12] OMG UML Press Release, 1997. Available on line at: http://www.omg.org/news/pr97/umlprimer.html

[13] "Modern Structured Analysis," Edward Yourdon, Yourdon Press, 1989.

[14]http://www.rational.com/uml/resources/documentation/media/ad970808_UML11_OCL.pdf.

[15] http://www.omg.org/docs/c4i/98-12-02.ppt

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center.........2
   8725 John J. Kingman Rd. STE 0944
   Ft. Belvoir, VA 22060-6218

2. Dudley Knox Library.........2
   Naval Postgraduate School
   411 Dyer Rd.
   Monterey, CA 93943-5000

3. Luqi.........1
   Naval Postgraduate School
   Computer Science Department
   411 Dyer Rd.
   Monterey, CA 93943-5000

4. Barbara McBride.........1
   Space and Naval Warfare Systems Command
   Code 051-1
   San Diego, CA

5. Thomas H. Augustine.........1
   Space and Naval Warfare Systems Center
   Code D4121
   53560 Hull Street
   San Diego, CA 92152-5001